(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 October 2002 (24.10.2002)

PCT

(10) International Publication Number
WO 02/084538 A1

(51) International Patent Classification[7]: G06F 17/50

(21) International Application Number: PCT/US01/11931

(22) International Filing Date: 11 April 2001 (11.04.2001)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: **MENTOR GRAPHICS CORPORATION** [US/US]; 8005 SW Boeckman Road, Wilsonville, OR 97070-7777 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **POCHAYEVETS, Olexandr** [UA/DE]; Berckhauserstrasse 42, 90409 Nuernberg (DE). **SIEGL, Johann** [DE/DE]; Georg-Simon-Ohm-Fachhochschule, Kesslerplatz, 90489 Nuernberg (DE). **EICHELE, Herbert** [DE/DE]; Georg-Simon-Ohm-Fachhochschule, Kesslerplatz, 90489 Nuernberg (DE).

(74) Agents: **DIEHL, Robert, A.** et al.; Columbia IP Law Group, PC, 10260 SW Greenburg Road, Suite 820, Portland, OR 97223 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 02/084538 A1

(54) Title: HDL PREPROCESSOR

(57) Abstract: The present invention describes a VHDL preprocessor. Configurable and flexible VHDL sources comprise statements that are replaceable by specific values dependent on design requirements.

HDL PREPROCESSOR

FIELD OF THE INVENTION

The present invention relates to a preprocessor for code written in a hardware description language, preferably VHDL. In particular, the present invention relates to methods, apparatuses and systems for configuring projects and/or IP cores written in a hardware description language.

BACKGROUND OF THE INVENTION

Since digital computer consisting of digital circuits are available engineers were looking for an appropriate form to describe such digital circuits by means of computers. The aim was to simulate circuits and to test their function before the circuits were transferred into hardware. Such form is, for example, HDL (**H**ardware **D**escription **L**anguage). The latest step in this respect is the development of VHDL (**V**ery high speed integrated circuits **H**ardware **D**escription **L**anguage). VHDL was standardized in IEEE 1076 in 1987, extended in 1993 (IEEE 1164), and further extended in 1999 in IEEE 1076.1, also known as VHDL-AMS (**A**nalog and **M**ixed **S**ignals). VHDL can be used for documentation, verification, and synthesis of large digital hardware designs. This is actually one of the key features of VHDL, since the same VHDL code can achieve all three of these goals, thus saving a lot of effort and reducing the introduction of errors between translating a specification into an implementation.

Usually a VHDL-Model consists of three parts: an entity, an architecture, and one or more configurations. Sometimes also packages are used and test benches must be written to test models.

An entity declaration in VHDL is a statement that defines the external specification of a circuit or sub-circuit. Using the information provided in an entity declaration (the port names and the data type and direction of each port), one has all the information needed to connect that portion of a circuit into

other, higher-level circuits, or to design input stimulus for testing purposes. The actual operation of the circuit, however, is not included in the entity declaration.

A very simple example is a NAND gate as shown in Fig. 1. At the top of the example is a library clause (`LIBRARY IEEE;`) and a use clause (`USE IEEE.std_logic_1164.ALL;`). This gives the entity access to all names declared within package STD_LOGIC_1164 in the library IEEE. The entity declaration includes the name of the entity and a set of generic and port declarations. GENERICS are constants passed into components, usually counts or delays. The PORT statement allows to define system I/O. A port may correspond to a pin on a IC, an edge connector on a board, etc. The following are signal types which can be declared in the port statement:

IN	Input to the system
OUT	Output from the system
INOUT	A bi-directional signal
BUFFER	A register attached to an output. (It is normally impossible to read an output, but a buffer allows this.)

The generic in the shown example is defined with type time because it is a delay value of 5 ns, although it is possible to pass in any VHDL type.

The second part of a VHDL-model is the architecture declaration. Every entity declaration must be accompanied by at least one corresponding architecture in order to describe how the model operates. VHDL allows to create more than one alternate architecture for each entity. An architecture declaration consists of zero or more declarations (of items such as intermediate signals, components that will be referenced in the architecture, local functions and procedures and constants) followed by a begin statement and a series of concurrent statements. The name of the architecture body is just an arbitrary label selected by the user. Within an architecture all statements are concurrent. There are two commonly used approaches for an

architecture description, structural and behavioral. Fig. 2 gives an example of how a complete system can be build hierarchically by combining structural and behavioral models.

A behavioral model is one which defines the behavior of a system, i.e., how a system acts. The used statements are PROCESS, WAIT, IF, CASE, FOR-LOOP. The NAND gate of Fig. 1 can be described in a behavioral model as follows:

```
architecture behavior of NAND is
begin
  c <= NOT (a AND b) after DELAY;
end behavior;
```

The architecture contains a concurrent signal assignment which describes the function of the design entity. The concurrent assignment executes, whenever one of the ports a or b changes its value. The order in which concurrent signals are written has no effect on their execution. The signal assignment from the example has a delay, that means that the signal on the left hand side is updated after the given delay.

As already mentioned, a structural model can be many levels deep, starting with primitive gates and building to describe a complete system. A RS-FlipFlop can be constructed from the simple NAND gate above, as can be seen in Fig. 3.

The FlipFlop is defined as two interconnected NAND instantiations. These are defined in separate VHDL code. This allows to use different levels in coding. For example, it would be possible to instantiate the RS-FF component to form a shift register. It is necessary that the ports in a component declaration match the ports in the entity declaration one for one. The names, order, mode and types of the ports to be used are defined in the component declaration. A

component has to be declared only once within an architecture, but may be instanced several times (two times in this example).

In this example the component NAND has two inputs (a and b) and an output (c). There are two instances of the NAND component in this architecture. The first line of the component instantiation statement gives this instance a name, `nand1`, and specifies that it is an instance of the component NAND. The second line describes how the component is connected to the rest of the design using the port map clause. The port map clause specifies what signals of the design to connect to the interface of the component in the same order as they are listed in the component declaration. The interface is specified in order as a, b, and then c, so this instance connects r to a, qb to b, and q to c. The second instance, named `nand2`, connects q to a, s to b, and qb to c of a different instance of the same NAND component.

The next type of design unit available in VHDL is called a configuration declaration. A configuration can be thought of as being roughly analogous to a parts list for a design. It specifies which architectures are to be bound to which entities. This allows to have different architectures bound to an entity statement and to change how components are connected in a design description at the time of simulation or synthesis. Configurations are optional, the VHDL standard specifies a set of rules that provides a default configuration, e. g. if more than one architecture for an entity is provided, the last architecture compiled will take precedence and will be bound to the entity.

A so-called package is used to collect commonly used declarations for use globally among different design units. It is identified by the PACKAGE keyword and can consist of two parts, a package declaration and an optional package body. Packages allow convenient ways of defining functions and constants which are used in more than one VHDL programs. They can be thought of as being a common storage area. Items defined within a package

can be made visible to any other design unit in the complete VHDL design. They also can be compiled into libraries for later use.

Packages act like an entity, i.e., they declare the interfaces to the functions and sub-programs found in the package body. Packages defining one or more deferred constants or containing declarations of sub-programs additionally require a package body. The relationship between a package and package body is similar to the relationship between an entity and its architecture. Like an architecture a package body must have the same name as its corresponding package declaration.

When modeling a circuit design, usually pre-designed models, for example VHDL models, available on the market are used (for example models for Fast Fourier Transform, Fast Cosine Transform etc.). These models are however predefined. This is to some extent disadvantageous as it is required that some characteristics can be flexibly configured, i.e., having variable parameters. The problem here is that a hardware description language project (e.g., VHDL-project) contains hardly configurable modules, for example:

```
GENERIC (width)
  IF in_adr = "XX..X" THEN
    out_d <= "XXXXXXX...XXX";   -- cos(2*Pi*I), I=0...n
--            <--- width --->
  ELSIF in_adr = "XX...X" THEN
    ...
END IF;
...
```

There is thus a demand that modules are configurable or flexible, such as models of CPUs or FFTs (Fast Fourier Transform).

Fig. 4 shows two conventional methods, i.e., "standard solution 1" and "standard solution 2" for providing such a flexibility in configuration. Both standard ways of VHDL project configuration require some external generation program (for example C) that creates a data file or directly a VHDL file. The standard of VHDL includes native configuration facilities expressed via "GENERIC", "GENERATE" and "AGGREGATE" directives but these do not satisfy all configuration needs: most sophisticated parts of VHDL code are to be configured via external software that prepares text files to be imported into VHDL source through the "textio" package. This is inconvenient and inefficient.

SUMMARY OF THE INVENTION

Thus, it is the object of the present invention to provide a more simple and convenient way of flexibly configuring modules written in a hardware description language, for example under VHDL. This object is achieved with the features of the claims.

The present invention provides a preprocessor for configuring projects and IP cores written in a hardware description language, and preferably written in VHDL. The present invention is based on the general idea that a configurable and thus flexible hardware description language source code, for example a VHDL source code, is obtained by first adding to the source statements that are then evaluated during the preprocessing operation and substituted by specific values depending on the specifically required configuration. Thus, a plurality of different specific cores can be obtained dependent on the requirements, and all unnecessary components are cancelled. These included statements are preferably LISP statements. The statements are preferably included surrounded by back single quotas ("`") in order to allow an identification of such configurable statements. The statements are a general description or terms representing parameters that are allowable at the particular location of the statement in the source code (for example the statement `` `wordwidth` `` being a general variable term for the range, e.g., 8 to 32 bits). In other words, the preprocessor according to the

present invention scans the source code for statements included in back single quotas ("`"), evaluates said statements, and replaces said general statements by calculated specific values dependent on the particular requirements. Hence, a single configurable and thus flexible source can be used for different applications and is adapted for each specific application by the present preprocessor.

The use of LISP statements in the present (e.g., VHDL) preprocessor is advantageous for several reasons. First, LISP is capable of performing as a self-modified program via its artificial intelligent functions like (mapcar). This feature provides a convenient way of evaluating the statements included into the VHDL source. Second, a LISP statement returns a single value after evaluation of the included statement. This value is then inserted into the VHDL source like a substitution of an included statement so that the VHDL source in its general form is transferred into a specific VHDL source dependent on the required configuration. All possible outputs that might be issued during the evaluation of the LISP statement are redirected as a prefix of the returned value.

Preferably, a test sequence of the IP core obtained with the VHDL preprocessor according to the present invention is automatically generated.

The preprocessor according to the present invention can be loaded into and operated in a general purpose computer or a dedicated computer, either of which includes both a processor and a memory, and is not limited in practice on any one particular type of computer.

The preprocessor according to the present invention has the following advantages. The preprocessor solves all configuration needs of a Hardware Description Language project, no additional external generation programs are required. Having the proposed preprocessor principle included into existent packages (for example, VHDL packages) vital problems of code configuration

with respect to flexibility will be solved. The present preprocessor can also be used as a generator of IP cores, which:

- will be optimized having no redundant units and switchers;
- will be compiled and mapped fast;
- will hide all architectural scalability from the end user;
- will have their particular test benches.

According to a first aspect of the present invention there is provided a method for configuring projects and/or IP cores written in a hardware description language, said method comprising the steps of:

providing source code of projects and/or IP cores to be configured;

including in said source code statements representing configurable parameters;

evaluating said included statements based on input data representing a specific configuration desired by a user for obtaining specific values for said configurable parameters based on said input data; and

replacing said statements by said calculated specific values.

An advantage of the method according to the first aspect is that existing sources, for example VHDL sources, can be modified in order for obtaining flexible and configurable sources.

According to a second aspect of the present invention, there is provided a method for obtaining configured projects and/or IP cores written in a hardware description language as desired by a user based on flexibly configurable projects and/or IP cores, said flexibly configurable projects and/or IP cores comprising source code having statements representing configurable parameters, said method comprising the steps of:

evaluating said included statements based on input data representing a specific configuration for obtaining specific values for said configurable parameters based on said input data; and

replacing said statements by said calculated specific values.

Preferably, the source code is written in Very High Speed Integrated Circuit Hardware Description Language (VHDL) source code. Preferably, said statements are LISP statements. According to a preferred embodiment, a test sequence of the specifically configured project and/or IP core is generated.

According to a third aspect of the present invention, there is provided an apparatus for configuring projects and/or IP cores written in a hardware description language, said apparatus comprising storage means for providing source code of projects and/or IP cores to be configured; means for including in said source code statements representing configurable parameters; evaluating means for evaluating said included statements based on input data representing a specific configuration desired by a user for obtaining specific values for said configurable parameters based on said input data; and computing means replacing said statements by said calculated specific values.

According to a fourth aspect of the present invention, there is provided an apparatus for obtaining configured projects and/or IP cores written in a hardware description language as desired by a user based on flexibly configurable projects and/or IP cores, said flexibly configurable projects and/or IP cores comprising source code having statements representing configurable parameters, said apparatus comprising valuating means for evaluating said included statements based on input data representing a specific configuration for obtaining specific values for said configurable parameters based on said input data; and computing means for replacing said statements by said calculated specific values.

According to a fifth aspect of the present invention, there is provided a system for configuring projects and/or IP cores written in a hardware description language, said system comprising a computer having a processor and a memory, said memory having stored therein a preprocessing program for (i) providing source code of projects and/or IP cores to be configured; (ii)

including in said source code statements representing configurable parameters; (iii) evaluating said included statements based on input data representing a specific configuration desired by a user for obtaining specific values for said configurable parameters based on said input data; and (iv) replacing said statements by said calculated specific values.

According to a sixth aspect of the present invention, there is provided a system for obtaining configured projects and/or IP cores written in a hardware description language as desired by a user based on flexibly configurable projects and/or IP cores, said flexibly configurable projects and/or IP cores comprising source code having statements representing configurable parameters, said system comprising a computer having a processor and a memory, said memory having stored therein a preprocessing program for (i) evaluating said included statements based on input data representing a specific configuration for obtaining specific values for said configurable parameters based on said input data; and (ii) replacing said statements by said calculated specific values.

According to a seventh aspect of the present invention there is provided a computer program comprising program code means for performing all the steps of any one of the methods of the first or second aspect when said program is run on a computer.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows the schematic and entity of a NAND gate;

Fig. 2 shows an example of a hierarchical VHDL-model;

Fig. 3 shows a structural description of a RS-FlipFlop;

Fig. 4 shows two conventional methods of VHDL project configuration;

Fig. 5 is a flow diagram showing the function (process_file) according to the present invention;

Fig. 6 is a flow diagram showing the function (process_hierarchy) according to the present invention;

Fig. 7 is a flow diagram showing the main function (VHDL_preprocessor) according to the present invention;

Fig. 8 shows the internal architecture of the VHDL preprocessor according to the present invention;

Fig. 9 shows the principle of the VHDL preprocessor according to the present invention;

Fig. 10 is a flow diagram showing the LISP subset runtime environment implemented according to the present invention;

Fig. 11 is a flow diagram showing the function lisp_parse() according to the present invention;

Fig. 12 is a flow diagram showing the function lisp_compile() according to the present invention;

Fig. 13 is a flow diagram showing the function lisp_link() according to the present invention;

Fig. 14 is a flow diagram showing the function lisp_run() according to the present invention; and

Fig. 15 is a schematic diagram showing the use of the VHDL preprocessor according to the invention for generating IP cores.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS AND EXAMPLES

The internal architecture of the VHDL preprocessor according to the present invention is shown in Fig. 8. The internal architecture of the VHDL preprocessor according to the present invention comprises three layers that provide flexibility and portability for various computer platforms. The top layer is implemented in LISP subset and comprises special statements that are included to VHDL project files. The second layer, i.e., the medium layer is also implemented in LISP subset and comprises the VHDL hierarchy preprocessing routine. Finally, at the low layer that is implemented in ANSI C the LISP subset runtime environment is provided. The implemented LISP subset runtime environment is described later.

Fig. 9 shows the principle of the VHDL preprocessor according to the present invention. Starting point are the VHDL/LISP project sources. "Source" represents the VHDL model including LISP statements, denoted in Fig. 9 with "unit.vhl". Every section that requires some modification is substituted by a LISP-like statement. Thereafter, the VHDL/LISP project sources are preprocessed using the present preprocessor. The parameters required for a specific configuration are input via a VHDL/LISP project configuration menu. Subsequently, all LISP entries in the target VHDL project (denoted with "unit.vhd") are evaluated.

Fig. 7 is a flow diagram showing the main function (VHDL_preprocessor) according to the present invention. Upon start of the main function (VHDL_preprocessor), the individual parameters, the location of the source, the destination of the target, the top level file and the test bench file are entered via an interactive menu. Subsequently, the function (process_hierarchy) is invoked for said top level VHDL file and the function (process_file) is invoked for said test bench file.

As shown in the flow diagram of Fig. 6, the function (process_hierarchy) requests from the user a file name, and then invokes the function (process_file) for a VHDL file (PASS1). Thereafter it looks through the already preprocessed VHDL file and recursively invokes itself for all other VHDL files referenced in "COMPONENT" native VHDL directives (PASS2). Usually it starts from the top level of a hierarchy and it also can start from any level of a hierarchy with the purpose of interactive iterative debugging. Such a sequence of the preprocessing (PASS1, PASS2) is important for the automatic configuration of the hierarchy of the target project sources. During PASS1 after conditional processing in the LISP statements required "COMPONENT" directives will be inserted to the preprocessed VHDL file that will define how the function (process_hierarchy) will dive recursively during PASS2.

The function (process_file) is shown in the flow diagram of Fig. 5. This function gets the contents of the file SOURCE/filename.vhl from the hard disk, checks it and substitutes in an iterative process all included LISP statements by their calculated values. However, in case an error occurs, the included LISP statement is replace with the error.

Example 1

The following example shows the function of the VHDL preprocessor with reference to an 8-bit adder. First, the IP core sources and test bench sources are provided. The following statements describe a general model of an adder that can be flexibly configured depending on the specific demand.

```
------------------------------------------------------
-- device.vhl ==> device.vhd                        --
------------------------------------------------------
--                                                   --
`(if use_rom
   (progn
     (outf "--          | a[2]             | b[2]                --\n" 0)
     (outf "--          |                  |                     --\n" 0)
     (outf "--   .------+------.   .------+------.                --\n" 0)
     (outf "--   |       u0     |   |       u1     |               --\n" 0)
     (outf "--   |   const ROM  |   |   const ROM  |               --\n" 0)
     (outf "--   |              |   |              |               --\n" 0)
     (outf "--   '------+------'   '------+------'               --\n" 0)
     (outf "--          |                  |                     --\n" 0)
     (outf "--          | opa[%02ld]" wordwidth)
     (outf "--          | opb[%02ld]              --\n" wordwidth)
     ""
   )
   (progn
     (outf "--          | a[%02ld]" wordwidth)
     (outf "--          | b[%02ld]              --\n" wordwidth)
     ""
   )
)`--          |                  |                     --
--    .-------+------------------+------.               --
--    |                                  |               --
--    |          A D D E R               |               --
--    |                                  |               --
--    '----------------+-----------------'               --
--                     |                                 --
--                     | s[`(str_fmt "%02ld" wordwidth)`]
--                                                       --
------------------------------------------------------
-- Generation time: `(time)`     --
------------------------------------------------------

LIBRARY IEEE;
```

```
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY device IS
  PORT(a,b : IN STD_LOGIC_VECTOR(`(if use_rom 1 (-- wordwidth))` DOWNTO 0)
       s: OUT STD_LOGIC_VECTOR(`(-- wordwidth)` DOWNTO 0));
END device;

ARCHITECTURE str OF device IS
`(if use_rom
     (progn
       (outf "  COMPONENT rom\n" nil)
       (outf "  PORT(in_adr : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);\n" nil)
       (outf "       out_d  : OUT STD_LOGIC_VECTOR(%ld DOWNTO 0));\n"
         (-- wordwidth))
       (outf "  END COMPONENT;\n" nil)
       (outf "  SIGNAL opa, opb : STD_LOGIC_VECTOR(%ld DOWNTO 0);\n"
         (-- wordwidth))
       ""
     )
     ""
)`BEGIN
`(if use_rom
   (progn
     (outf "  u0: rom\n" nil)
     (outf "      PORT MAP(in_adr => a, out_d => opa);\n" nil)
     (outf "  u1: rom\n" nil)
     "      PORT MAP(in_adr => b, out_d => opb);\n"
   )
   ""
)`  s <= `(if use_rom "op" "")`a + `(if use_rom "op" "")`b;
END str;
```

The ROM for the adder is described as follows:

```
-------------------------------------------------
-- rom.vhl ==> rom.vhd                          --
-------------------------------------------------
--                                               --
--              .-------------.                  --
--  in_adr[2]   |             |  out_d[`(str_fmt "%02ld" wordwidth)`]
--  >----------+  const ROM  +---------->        --
--              |             |                  --
--              '-------------'                  --
--                                               --
-------------------------------------------------
-- Generation time: `(time)`     --
-------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY rom IS
  PORT(in_adr : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
       out_d : OUT STD_LOGIC_VECTOR(`(-- wordwidth)` DOWNTO 0));
```

```
END rom;

ARCHITECTURE str OF rom IS
BEGIN
  run: PROCESS(in_adr)
  BEGIN
`(progn
   (defun coeff_gen
     (progn
       (setq wordwidth $1)
       (setq i $2)
       (setq n $3)
       (setq ret_line "\"0")
       (setq weight 0)
       (for j 2 1 wordwidth (progn
         (setq weight (<< weight 1))
         (setq weight (++ weight))
       ))
       (setq coeff (ival (*. weight (cos (/. (*. (pi) (++ (* 2 i))) (* 2
n)))))))
       (setq j (<< 1 (- wordwidth 2)))
       (while (>= j 1) (progn
         (setq ret_line (cat ret_line (if (> (/ coeff j) 0) "1" "0")))
         (setq coeff (% coeff j))
         (setq j (>> j 1))
       ))
       (cat (cat ret_line "\";  -- R") (cat (str i) (cat "(" (cat (str n)
         ")")))))
   )
  )
  (outf "    IF in_adr = "00" THEN\n" 0)
  (outf "      out_d <= %s\n" (coeff_gen wordwidth 0 8))
  (outf "    ELSIF in_adr = "01" THEN\n" 0)
  (outf "      out_d <= %s\n" (coeff_gen wordwidth 1 8))
  (outf "    ELSIF in_adr = "10" THEN\n" 0)
  (outf "      out_d <= %s\n" (coeff_gen wordwidth 3 8))
  (outf "    ELSIF in_adr = "11" THEN\n" 0)
  (outf "      out_d <= %s\n" (coeff_gen wordwidth 2 8))
  "    END IF;\n"
 )`  END PROCESS run;
END str;
```

The simplest test bench script is described as follows:

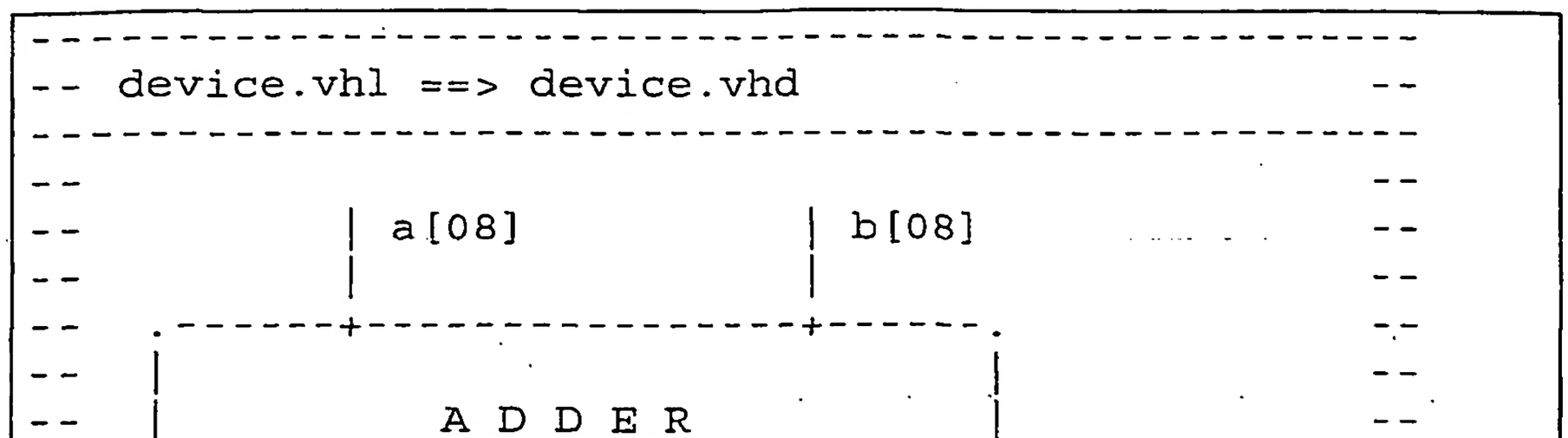
```
# devtst.dl ==> devtst.do

radix dec
wave *
`(progn
   (if use_rom
     (setq weight 4)
     (progn
       (setq weight 0)
       (for i 2 1 wordwidth (progn
         (setq weight (<< weight 1))
         (setq weight (++ weight))
       ))
     )
```

```
      )
      (for i 1 1 10 (progn
        (outf "\nforce a %ld\n" (irnd weight))
        (outf "force b %ld\n" (irnd weight))
        (outf "run 1000\n" nil)
      ))
      ""
   )`
  # <EOF>
```

The present VHDL preprocessor assumes that configurable VHDL sources include special statements surrounded by back single quotas "`". These particular statements are evaluated during the preprocessing of the VHDL sources, and are substituted by their specific calculated values. For example, the statement `(if use_rom 1 (-- wordwidth))` used in the above IP core for an adder is replaced by "7" as shown in the following example of an 8-bit adder without ROMs. Statements that are not required for the specific application/configuration are canceled. In case a 8-bit version of the adder without ROMs is considered, the data as shown in the following are entered via a configuration menu by the user:

```
VHDL PreProcessor (IP Core Generator) !

*          Device IP Generator          *
*          Configuration  Menu          *

  0.                 Wordwidth: `8'
  1.                   Use ROM: "No"
  2.  Source file directory: "SOURCE/"
  3.  Target file directory: "TARGET/"
  4.           TopLevel file: "device"
  5.         TestBench file: "devtst"
  G. Generate the Device.
  Q. Quit the program.

Enter choice: G
SOURCE/device ==> TARGET/device
SOURCE/devtst ==> TARGET/devtst
```

The final, i.e., generated IP core for the 8-bit version without ROMs has the following form:




```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY device IS
  PORT(a,b : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
       s: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END device;

ARCHITECTURE str OF device IS
BEGIN
  s <= a + b;
END str;
```

The simplest test bench script is as follows:

```
# devtst.dl ==> devtst.do

radix dec
wave *

force a 65
force b 22
run 1000

force a 39
force b 67
run 1000
```

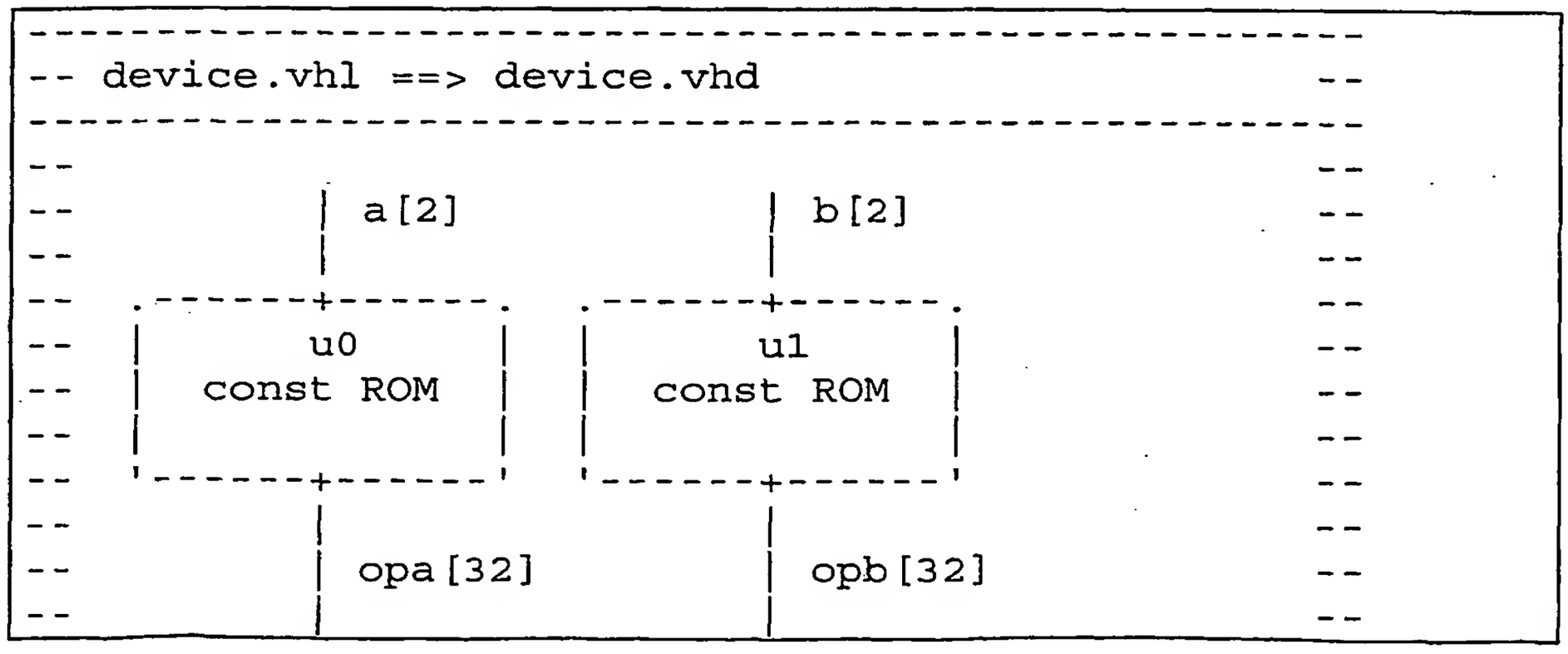
```
force a 120
force b 21
run 1000

force a 89
force b 28
run 1000

force a 62
force b 15
run 1000

force a 10
force b 49
run 1000

force a 35
force b 46
run 1000

force a 124
force b 67
run 1000

force a 97
force b 82
run 1000

force a 97
force b 99
run 1000

# <EOF>
```

Example 2:

Based on the same general model shown above, a specific model for a 32-bit version with ROMs can be obtained according to the VHDL-preprocessor of the present invention by entering the following parameters:

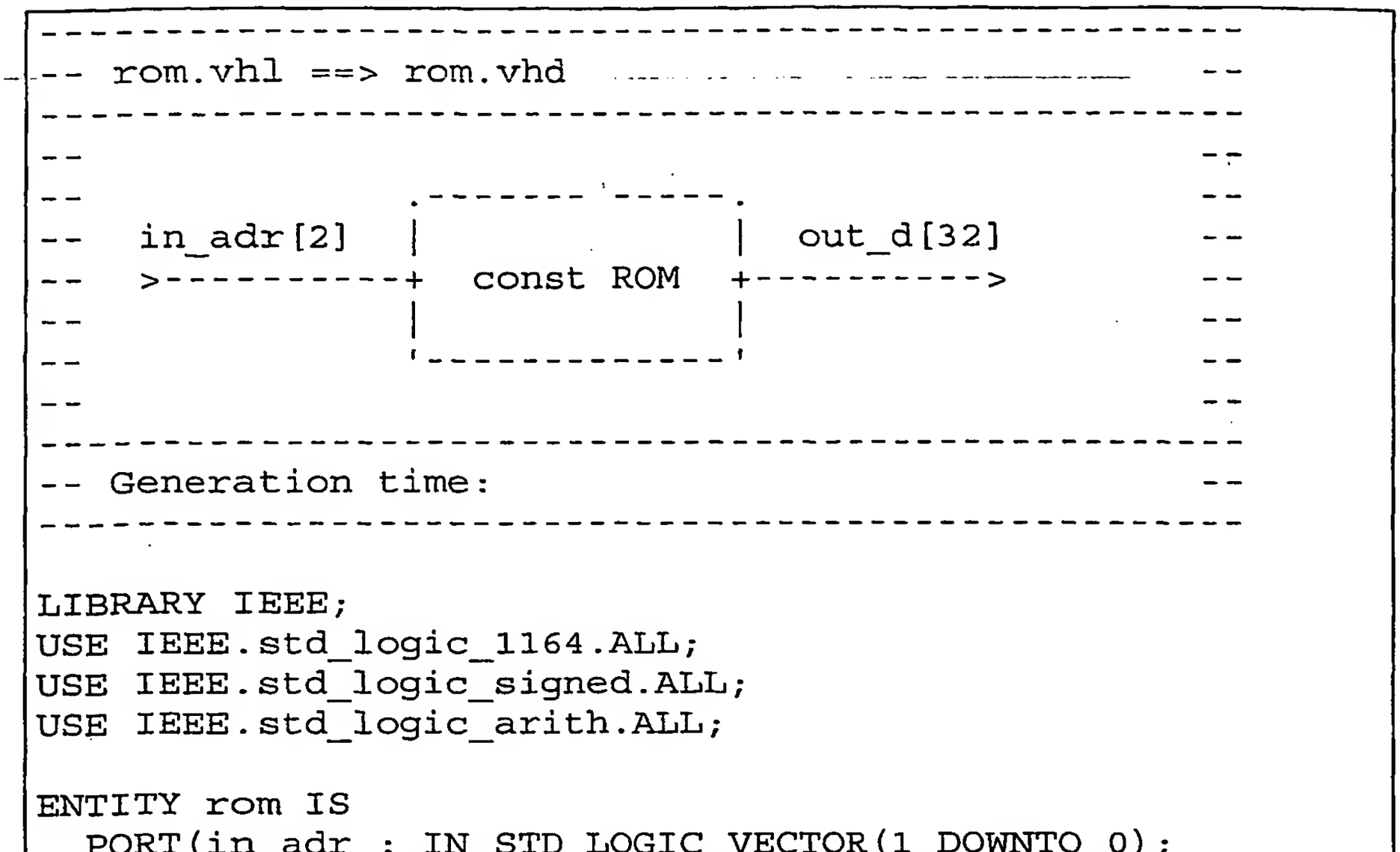
```
VHDL PreProcessor (IP Core Generator) !

*          Device IP Generator          *
*          Configuration  Menu          *

  0.                Wordwidth: `32'
  1.                 Use ROM: "Yes"
  2.  Source file directory: "SOURCE/"
  3.  Target file directory: "TARGET/"
  4.           TopLevel file: "device"
  5.         TestBench file: "devtst"
  G. Generate the Device.
  Q. Quit the program.

Enter choice: G
SOURCE/device ==> TARGET/device
  SOURCE/rom ==> TARGET/rom

SOURCE/devtst ==> TARGET/devtst
```

The generated specific IP core for such a 32-bit adder then has the following form:



```
--    .------+----------------+------.                    --
--    |                                |                    --
--    |            A D D E R          |                    --
--    |                                |                    --
--    '--------------+-----------------'                    --
--                   |                                      --
--                   | s[32]                                --
--                                                          --
------------------------------------------------------------
-- Generation time:                                        --
------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY device IS
  PORT(a,b : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
       s: OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END device;

ARCHITECTURE str OF device IS
  COMPONENT rom
  PORT(in_adr : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
       out_d  : OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
  END COMPONENT;
  SIGNAL opa, opb : STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
  u0: rom
      PORT MAP(in_adr => a, out_d => opa);
  u1: rom
      PORT MAP(in_adr => b, out_d => opb);
  s <= opa + opb;
END str;
```

The model of the associated ROMs is as follows:




```
-- Generation time:
--------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY rom IS
  PORT(in_adr : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
       out_d : OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END rom;

ARCHITECTURE str OF rom IS
BEGIN
  run: PROCESS(in_adr)
  BEGIN
    IF in_adr = "00" THEN
      out_d <= "01111101100010100101111100111110";   --
R0(8)
    ELSIF in_adr = "01" THEN
      out_d <= "01101010011011011001100010100011";   --
R1(8)
    ELSIF in_adr = "10" THEN
      out_d <= "00011000111110001011100000111100";   --
R3(8)
    ELSIF in_adr = "11" THEN
      out_d <= "01000111000111001110110011100110";   --
R2(8)
    END IF;
  END PROCESS run;
END str;
```

The simplest test bench script is as follows:

```
# devtst.dl ==> devtst.do

radix dec
wave *

force a 2
force b 0
run 1000

force a 1
force b 2
run 1000

force a 3
force b 0
run 1000

force a 2
force b 0
run 1000

force a 1
force b 0
run 1000

force a 0
force b 1
run 1000

force a 1
force b 1
run 1000

force a 3
force b 2
run 1000

force a 3
force b 2
run 1000

force a 3
force b 3
run 1000

# <EOF>
```

Example 3

The following example shows the function of the inventive VHDL preprocessor as a generator of the Fast Cosine Transform (FCT) processor cores which can be customized via a configuration menu. The parameters of the configuration menu and examples of generated cores are enumerated below.

- FCT word width: [2..32]bits
- FCT size0: [2,4,8,16,32,64,128,256] points for the first dimension
- FCT size1: [2,4,8,16,32,64,128,256] points for the second dimension
- Dimensioning: 1D for one-dimensional or 2D for two-dimensional transform
- Test number sequences: "Yes" to generate or "No" not to generate test set

Processor	Clock[ns]	LCs[n(%)]	RAM [bit]	FPGA
1D FCT 8bit 8	32	1971(16%)	384	ALTERA FLEX10K
1D FCT 16bit 8	48	6113(50%)	0	ALTERA FLEX10K
1D FCT 24bit 8	64	10311(84%)	0	ALTERA FLEX10K
2D FCT 8bit 8x8	64	5508(45%)	1024	ALTERA FLEX10K
2D FCT 10bit 8x8	64	7032(57%)	1280	ALTERA FLEX10K
2D FCT 12bit 8x8	80	8761(72%)	1536	ALTERA FLEX10K

In comparison to standard LISP, the LISP subset runtime environment implemented in the present invention and shown in Fig. 10 has the following features:

- Tiny size of the LISP kernel (120KB for CISC (Intel Pentium processor), 330KB for RISC (SPARC or MIPS processors)) due to minimal set of functions and optimized internal architecture.
- The LISP kernel can be easily embedded into other software systems like a CAD or a simulation suite due to separate functions lisp_parse() (see Fig. 11), lisp_compile() (see Fig. 12), lisp_link() (see Fig. 13), and lisp_run() (see Fig. 14).

- Runtime standard output can be redirected as a result of evaluation.

The implemented LISP subset has the traditional prefix form and supports integer, float, string, list and tree data types. The following functions are chosen according to the invention as a minimal set:

Category	Functions
Complex	(progn)
User Defined	(defun)
Variable/array creator/evaluator	(setq), (arsetq), (index)
Input/output, file access	(accept), (scan_console), (outf), (file_create), (file_open), (file_write), (file_read), (file_close), (file_remove)
Conditional execution	(if)
Iteration execution	(while), (for)
Built-in comparison and logical	("equal"/"=="), ("notequal"/"!="), ("less"/"<"), ("more"/">"), ("lessorequal"/"<="), ("moreorequal"/">="), ("and"/"&&"), ("or"/"||"), ("not"/"!")
Built-in integer	(ival), (indices), (irnd), ("iadd"/"+"), ("isub"/"-"), ("imul"/"*"), ("idiv"/"/"), ("ima"/"*+"), ("imod"/"%"), ("iincr"/"++"), ("idecr"/"--"), ("ineg"/"0-"), (iabs), ("iand"/"&"), ("ior"/"|"), ("ixor"/"^"), ("inot"/"~"), ("ishr"/">>"), ("ishl"/"<<")
Built-in float	(fval), (frnd), ("fadd"/"+."), ("fsub"/"-."), ("fmul"/"*."), ("fdiv"/"/."), ("fma"/"*+."), (fabs), ("fint"/"int"), ("fround"/"round"), ("fcos"/"cos"), ("fsin"/"sin"), ("fcas"/"cas"), ("fatn"/"atn"), ("fexp"/"exp"), ("fpow"/"pow"/"^."/"**"), ("fln"/"ln"), ("fsqrt"/"sqrt"/"sqr")
Built-in string	(str), (chr), (asc), (type), (dump_i2s), (dump_f2s), (dump_s2i), (dump_s2f), (notempty), (len), (at), (rat), (cat), (space), (replicate), (left), (leftr), (right), (rightl),

	(substr), (strtran), (str_raw), (str_unraw), (str_dump), (str_fmt), (ltrim), (rtrim), (alltrim), (pack), (head), (tail), (upper), (lower), (rev), (padl), (padr), (padc), (time), (getenv)
Built-in constant	(ee), (gamma), (phi), (pi), (prn_integer_fmt), (prn_float_fmt), (prn_string_fmt), (reinit_terminal), (term_type), (lines_term), (columns_term), (clrscr_term), (reverse_term), (blink_term), (bold_term), (normal_term), (hidecursor_term), (showcursor_term), (gotocursor_term), (gotocursor1_term), (version_fstlisp), (version_termcap), (version_strglib), (version_mempool), (compiled_on), (compiled_by)
Built-in artificial intelligence	(mapcar)

Fig. 15 is a schematic diagram showing the use of the VHDL preprocessor according to the invention for generating IP cores. The IP cores obtained by the present preprocessor are optimized having no redundant units and switchers, are prepared to be compiled and mapped fast, hide all architectural scalability from the end user and have their particular test benches.

CLAIMS

1. A method for configuring projects and/or IP cores written in a hardware description language, said method comprising the steps of:
providing source code of projects and/or IP cores to be configured;
including in said source code statements representing configurable parameters;
evaluating said included statements based on input data representing a specific configuration desired by a user for obtaining specific values for said configurable parameters based on said input data; and
replacing said statements by said calculated specific values.

2. A method for obtaining configured projects and/or IP cores written in a hardware description language as desired by a user based on flexibly configurable projects and/or IP cores, said flexibly configurable projects and/or IP cores comprising source code having statements representing configurable parameters, said method comprising the steps of:
evaluating said included statements based on input data representing a specific configuration for obtaining specific values for said configurable parameters based on said input data; and
replacing said statements by said calculated specific values.

3. The method of claim 1 or 2, wherein the source code is written in Very High Speed Integrated Circuit Hardware Description Language (VHDL) source code.

4. The method of any of claims 1 to 3, wherein said statements are LISP statements.

5. The method of any of claims 1 to 4 , wherein said input data are input by a user.

6. The method of any of claims 1 to 5 , further comprising the step of generating a test sequence of the specifically configured project and/or IP core.

7. Apparatus for configuring projects and/or IP cores written in a hardware description language, said apparatus comprising:

storage means for providing source code of projects and/or IP cores to be configured;

means for including in said source code statements representing configurable parameters;

evaluating means for evaluating said included statements based on input data representing a specific configuration desired by a user for obtaining specific values for said configurable parameters based on said input data; and

computing means replacing said statements by said calculated specific values.

8. Apparatus for obtaining configured projects and/or IP cores written in a hardware description language as desired by a user based on flexibly configurable projects and/or IP cores, said flexibly configurable projects and/or IP cores comprising source code having statements representing configurable parameters, said apparatus comprising:

valuating means for evaluating said included statements based on input data representing a specific configuration for obtaining specific values for said configurable parameters based on said input data; and

computing means for replacing said statements by said calculated specific values.

9. The apparatus of claim 7 or 8, wherein the source code is written in Very High Speed Integrated Circuit Hardware Description Language (VHDL) source code.

10. The apparatus of any of claims 7 to 9, wherein said statements are LISP statements.

11. The apparatus of any of claims 7 to 10 , further comprising an input terminal for inputting said input data by a user.

12. The apparatus of any of claims 7 to 11 , further comprising generating means for generating a test sequence of the specifically configured project and/or IP core.

13. System for configuring projects and/or IP cores written in a hardware description language, said system comprising a computer having a processor and a memory, said memory having stored therein a preprocessing program for (i) providing source code of projects and/or IP cores to be configured; (ii) including in said source code statements representing configurable parameters; (iii) evaluating said included statements based on input data representing a specific configuration desired by a user for obtaining specific values for said configurable parameters based on said input data; and (iv) replacing said statements by said calculated specific values.

14. System for obtaining configured projects and/or IP cores written in a hardware description language as desired by a user based on flexibly configurable projects and/or IP cores, said flexibly configurable projects and/or IP cores comprising source code having statements representing configurable parameters, said system comprising a computer having a processor and a memory, said memory having stored therein a preprocessing program for (i) evaluating said included statements based on input data representing a specific configuration for obtaining specific values for said configurable parameters based on said input data; and (ii) replacing said statements by said calculated specific values.

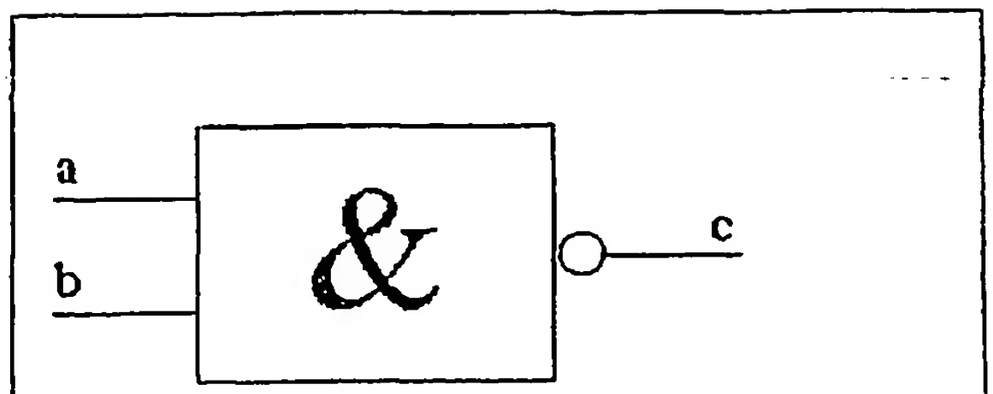
15. The system of claim 13 or 14, wherein the code is written in Very High Speed Integrated Circuit Hardware Description Language (VHDL) source code.

16. The system of any of claims 13 to 15, wherein said statements are LISP statements.

17. The system of any of claims 13 to 16 , further comprising an input terminal for inputting said input data by a user.

18. The system of any of claims 13 to 17 , further comprising generating means for generating a test sequence of the specifically configured project and/or IP core.

19. A computer program comprising program code means for performing all the steps of any one of the claims 1 to 6 when said program is run on a computer.




```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY nand IS
  GENERIC (delay  : = 5 ns);
  PORT (a         : IN  std_logic;
        b         : IN  std_logic;
        c         : OUT std_logic;
END nand;
```

Fig. 1

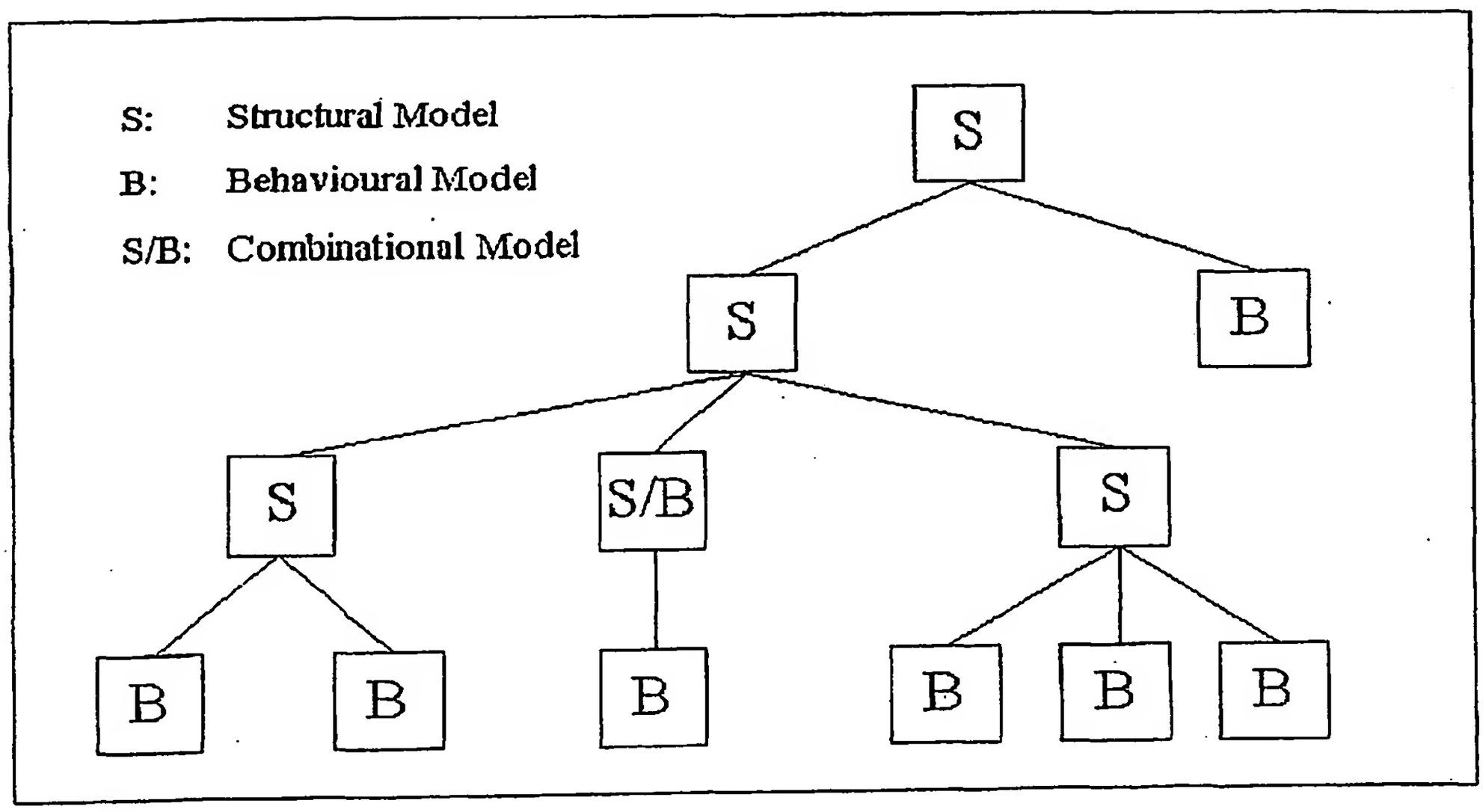


Fig. 2

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY rsff IS
      PORT (r    : IN  std_logic;
            s    : IN  std_logic;
            q    : OUT std_logic;
            qb   : OUT std_logic);
END rsff;
ARCHITECTURE struct OF rsff IS
      COMPONENT nand
            GENERIC (delay  : = 5 ns);
            PORT (a         : IN  std_logic;
                  b         : IN  std_logic;
                  c         : OUT std_logic;
      END COMPONENT;
BEGIN
      nand1: nand
            GENERIC MAP (5 ns);
            PORT MAP (r, qb, q);
      nand2: nand
            GENERIC MAP (5 ns);
            PORT MAP (q, s, qb);
END struct;
```

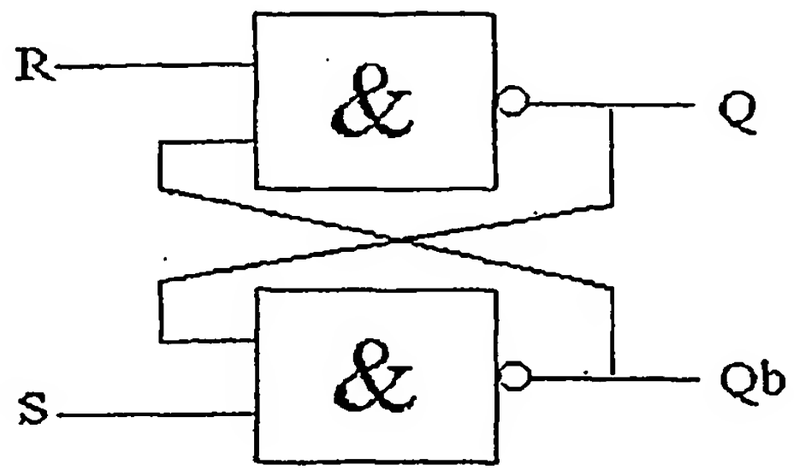


Fig. 3

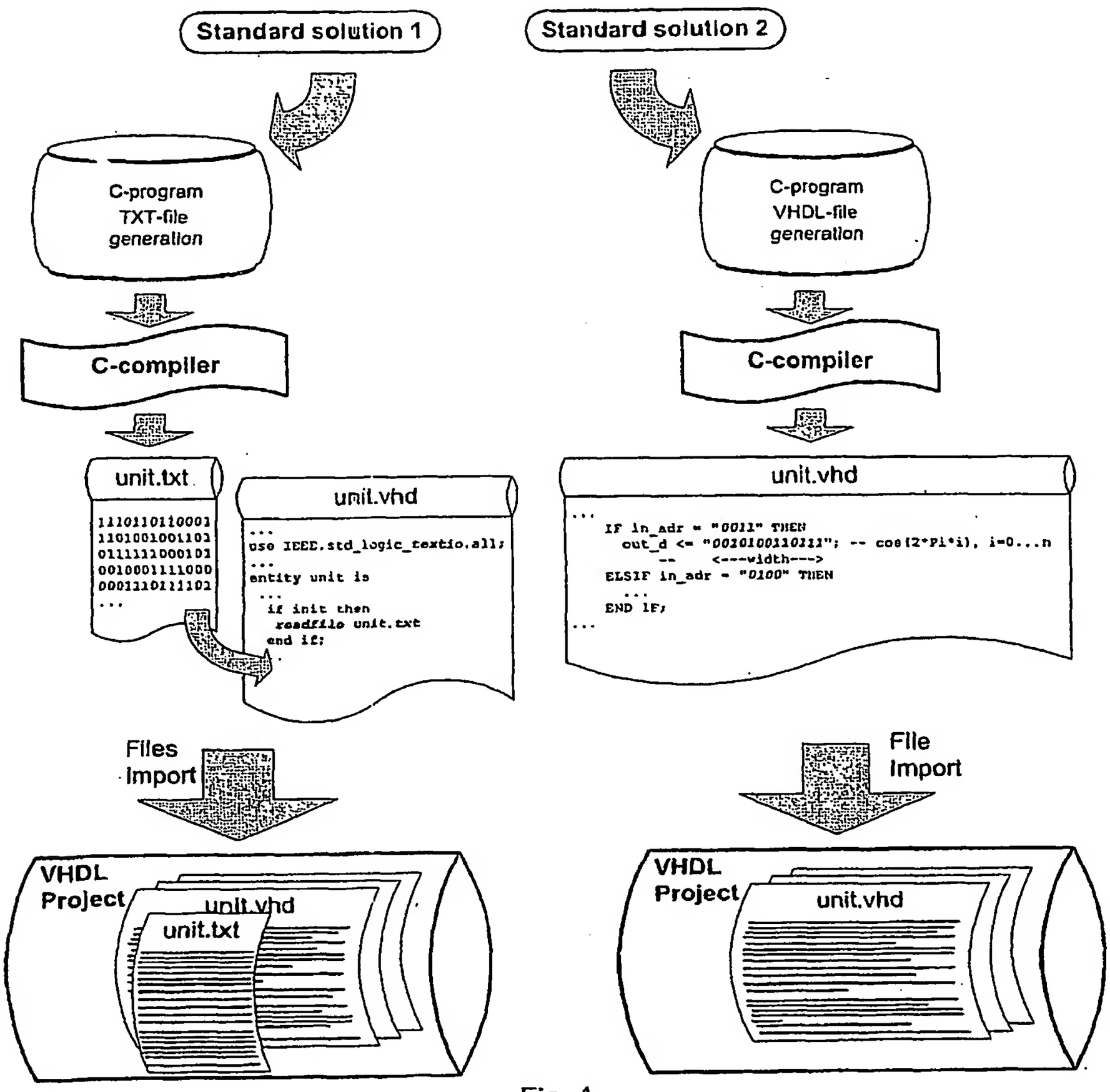


Fig. 4

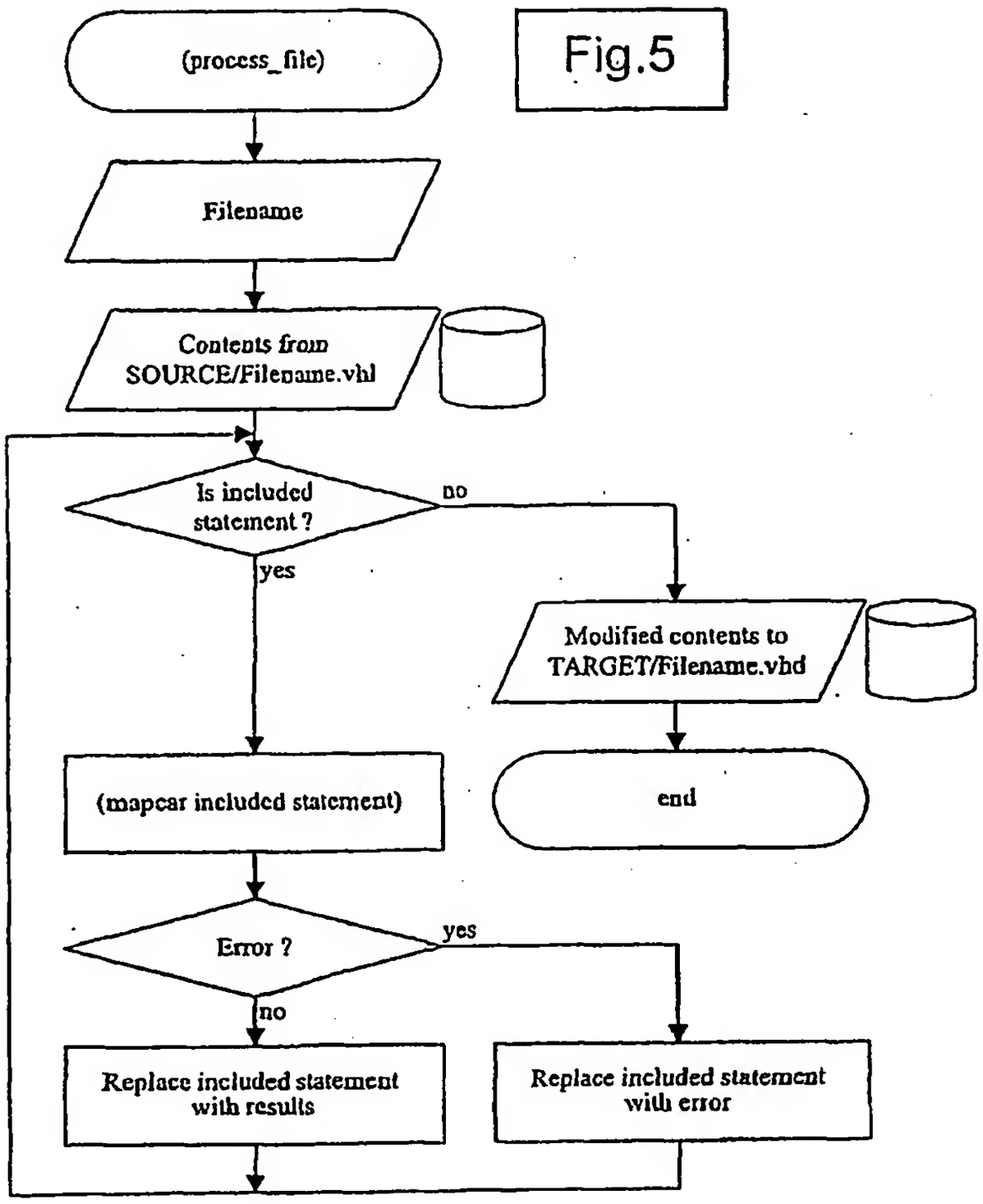


Fig.5

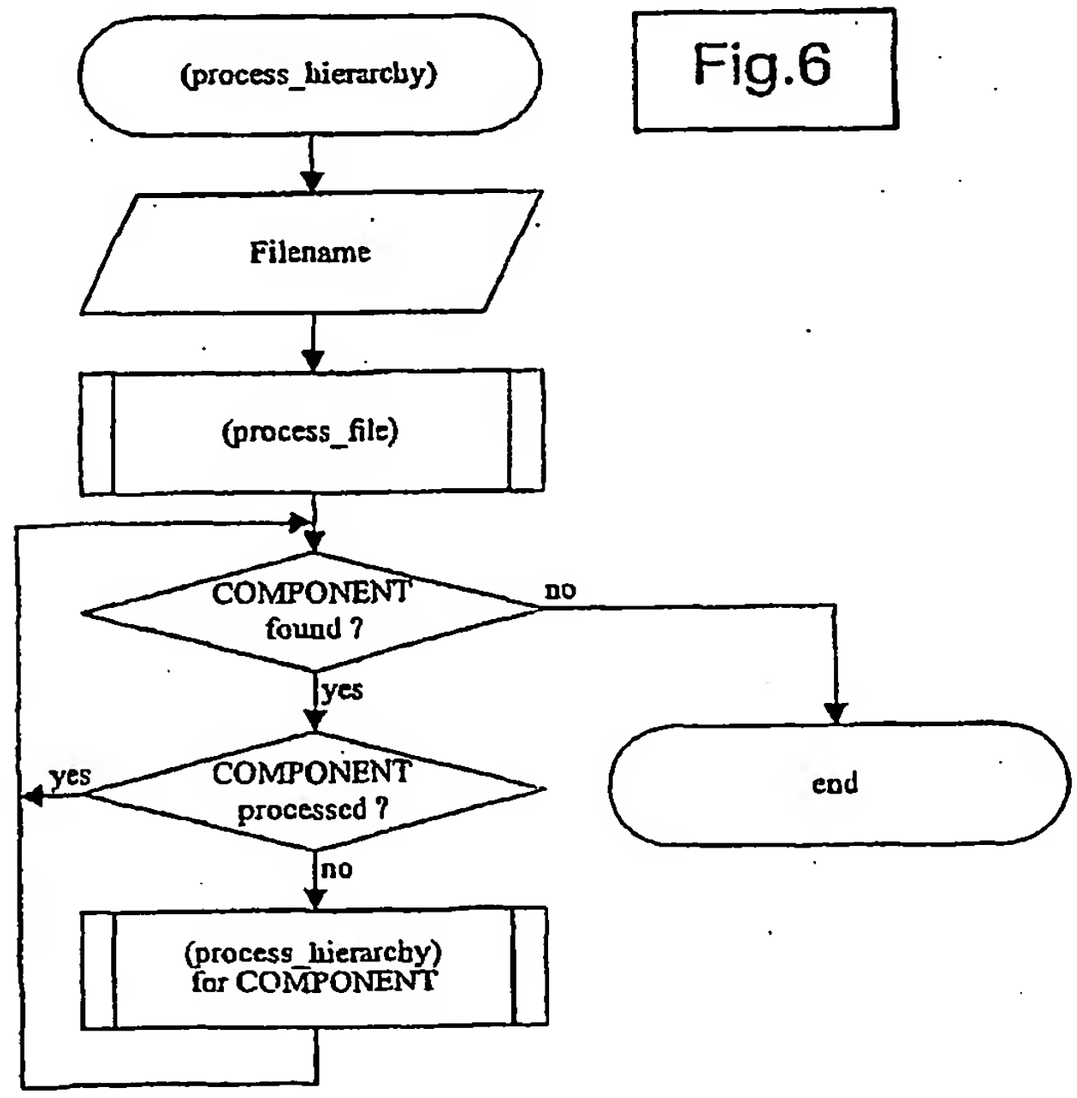


Fig.6

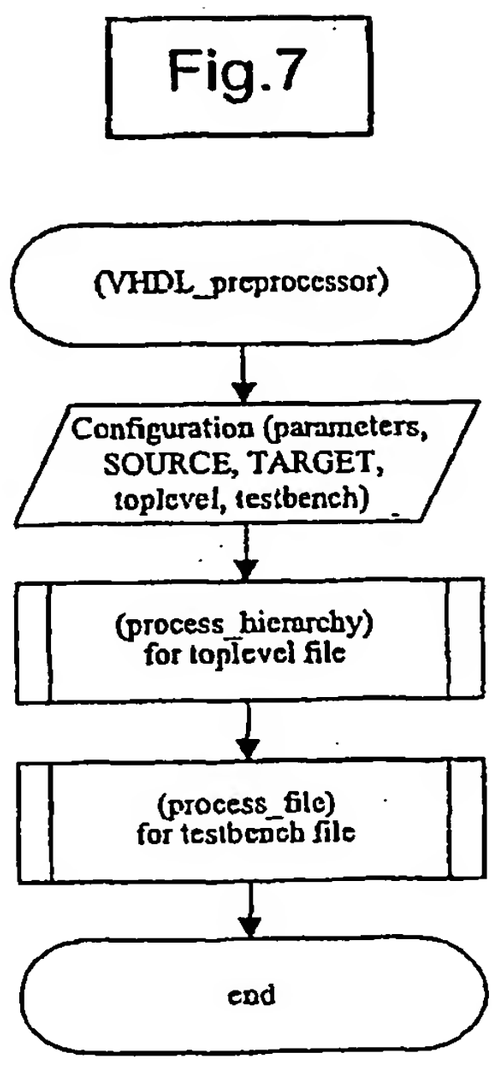


Fig.7

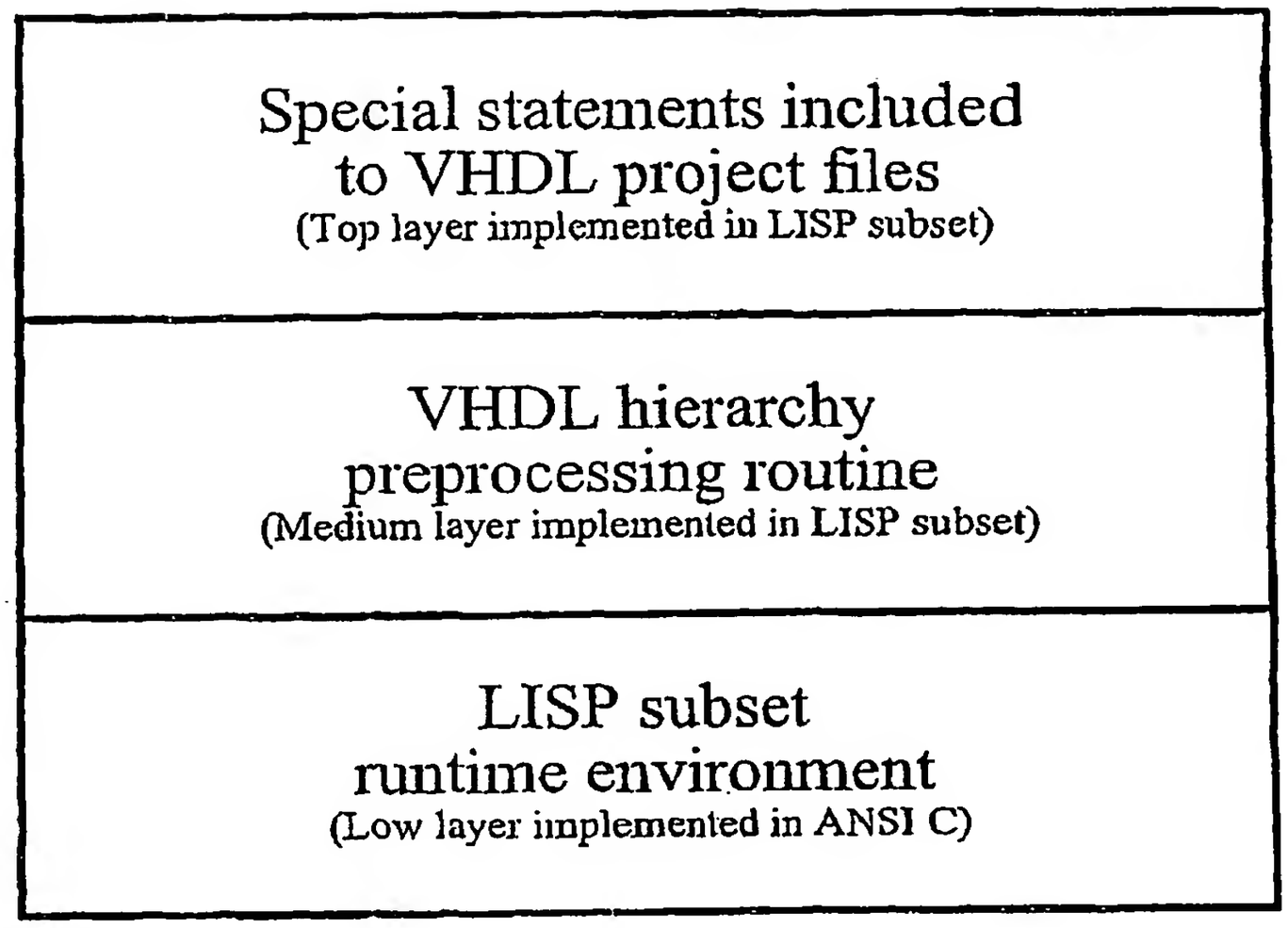


Fig. 8

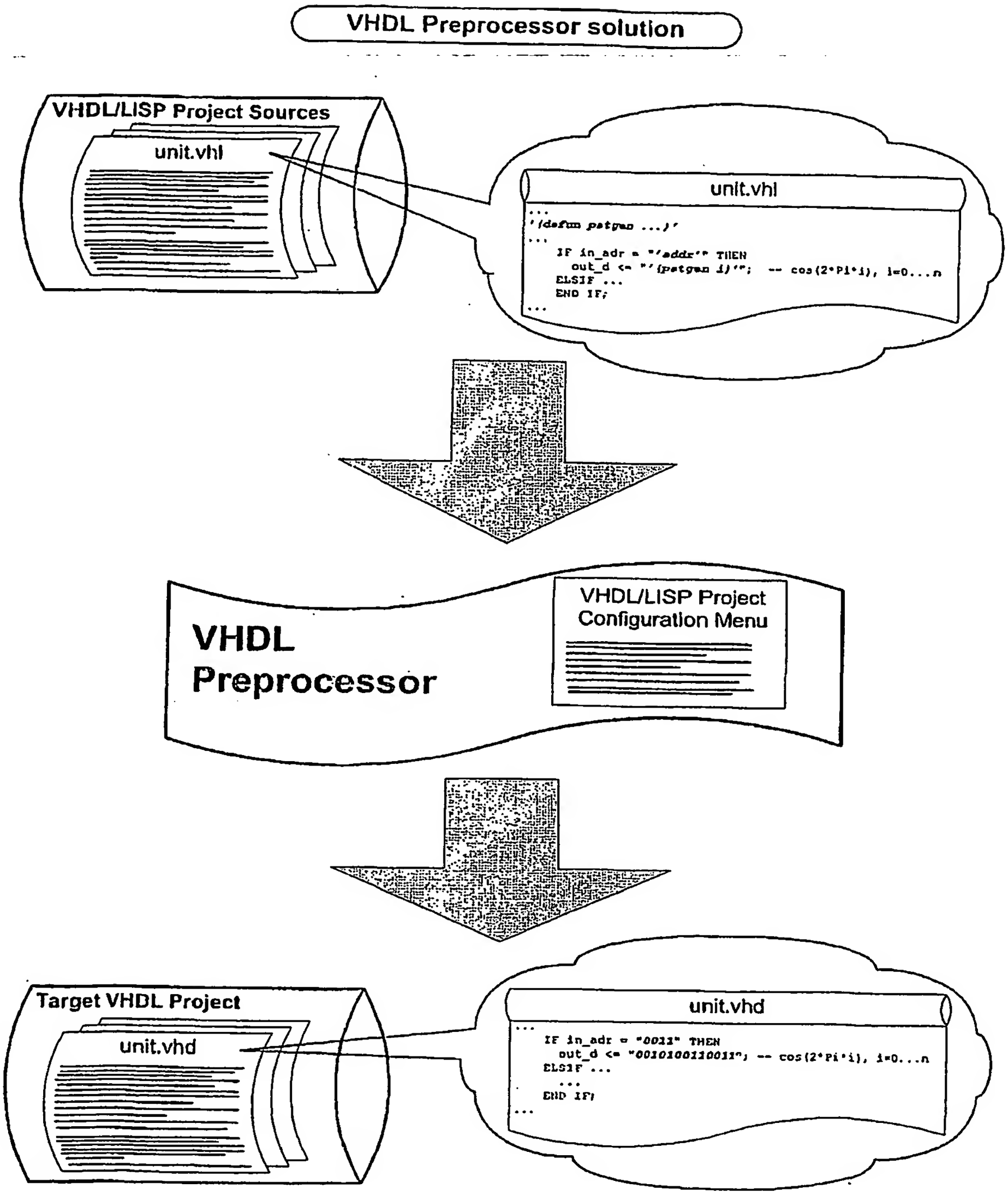

VHDL Preprocessor solution
VHDL/LISP Project Sources
unit.vhl
unit.vhl
...
'(defun patgen ...)'
...
IF in_adr = "'addr'" THEN
out_d <= "'(patgen i)'"; -- cos(2*PI*i), i=0...n
ELSIF ...
END IF;
...
VHDL
Preprocessor
VHDL/LISP Project
Configuration Menu
Target VHDL Project
unit.vhd
unit.vhd
...
IF in_adr = "0011" THEN
out_d <= "0010100110011"; -- cos(2*Pi*i), i=0...n
ELSIF ...
...
END IF;
...

Fig. 9

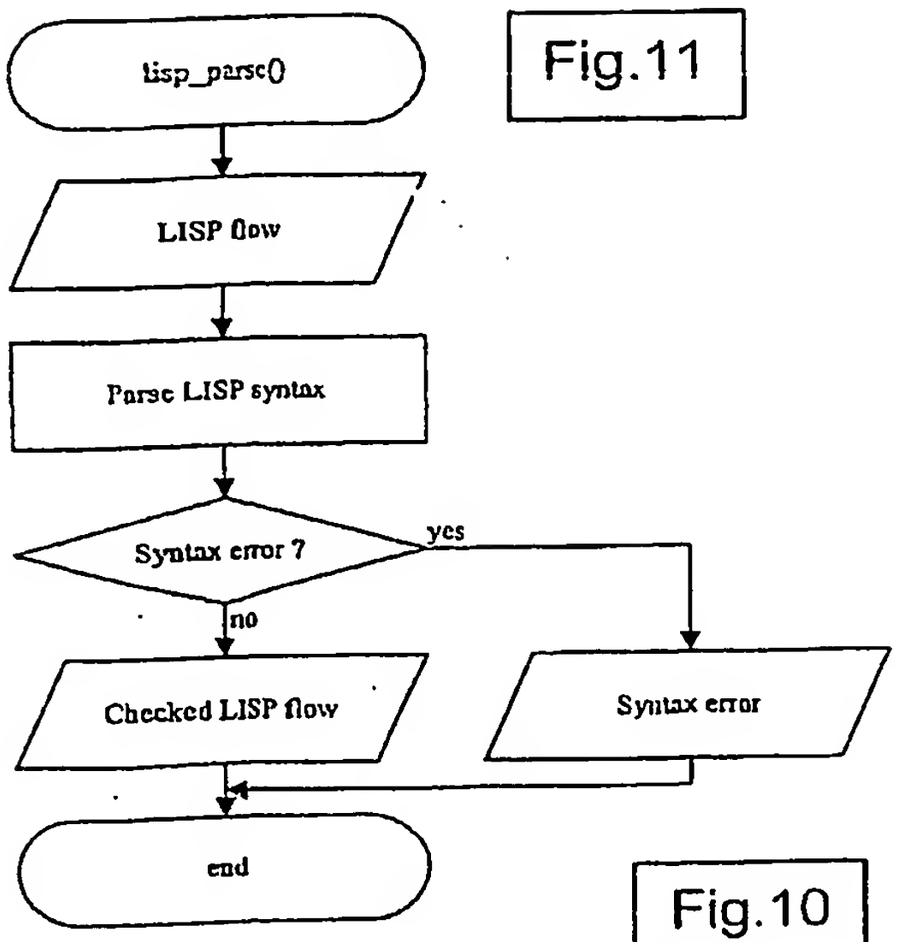


Fig.11

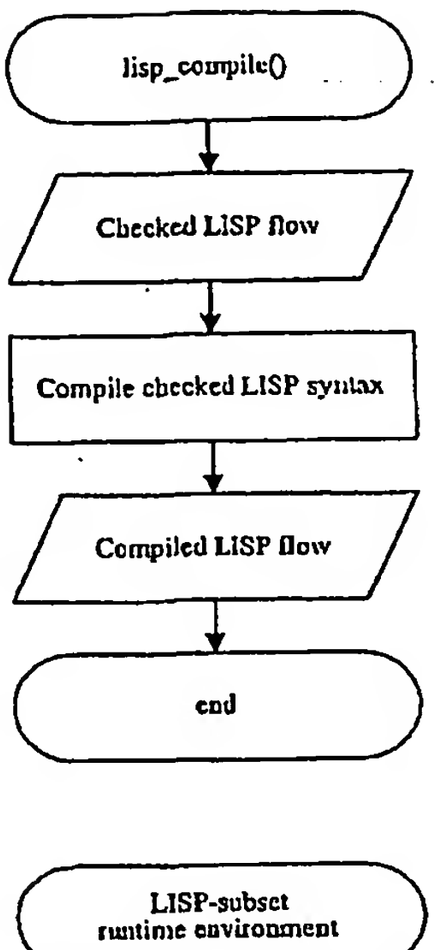


Fig.12

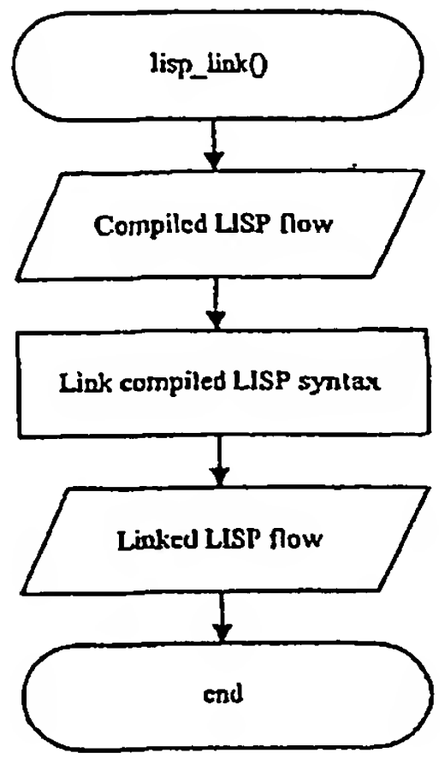


Fig.13

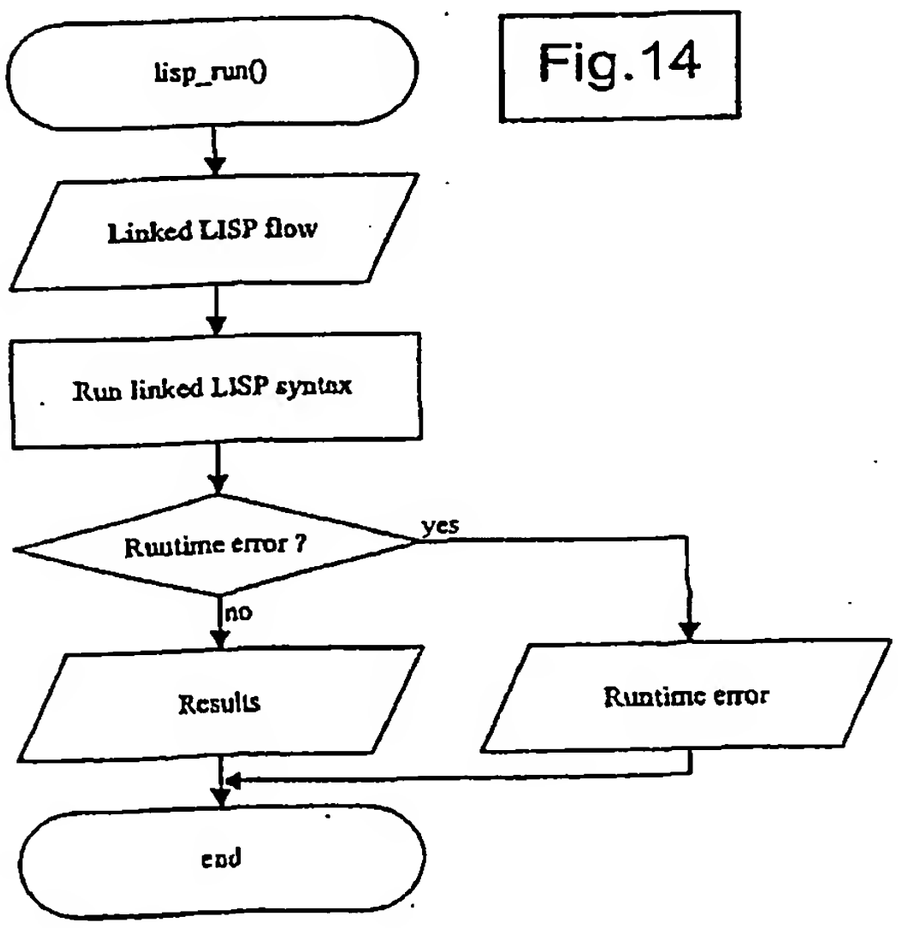


Fig.10

Fig.14

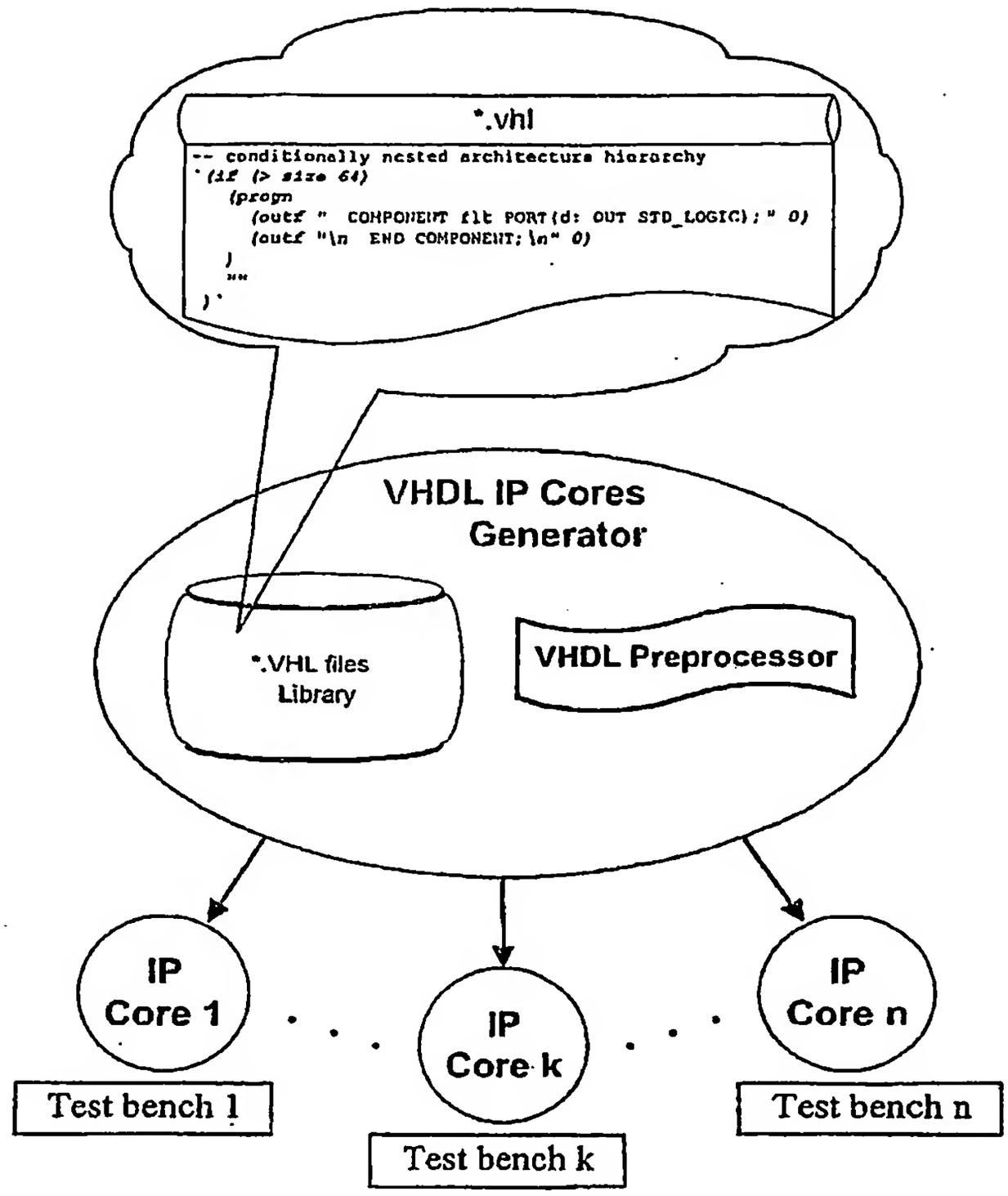


Fig. 15

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/11931

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F17/50

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
INSPEC, EPO-Internal, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	SEZER S ET AL: "VHDL package GUI" IEE SEMINAR ON INTELLECTUAL PROPERTY (REF. NO.00/082), IEE SEMINAR ON INTELLECTUAL PROPERTY, EDINBURGH, UK, 6 JULY 2000, pages 5/1-5, XP001023908 2000, London, UK, IEE, UK paragraphs '0004!,'0005! figures 1,2 --- -/--	1-19

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
20 December 2001	16/01/2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Guingale, A

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/11931

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	BERGAMASCHI R ET AL: "Coral-automating the design of systems-on-chip using cores" PROCEEDINGS OF THE IEEE 2000 CUSTOM INTEGRATED CIRCUITS CONFERENCE (CAT. NO.00CH37044), PROCEEDINGS OF THE IEEE 2000 CUSTOM INTEGRATED CIRCUITS CONFERENCE, ORLANDO, FL, USA, 21-24 MAY 2000, pages 109-112, XP002186200 2000, Piscataway, NJ, USA, IEEE, USA ISBN: 0-7803-5809-0 paragraphs '002A!,'002F! figure 1 ---	1-19
X	US 6 120 549 A (KELEM STEVEN H ET AL) 19 September 2000 (2000-09-19) column 2, line 15 - line 26 column 2, line 46 -column 5, line 23 figure 2 ---	1-19
X	EP 0 834 823 A (VLSI TECHNOLOGY INC) 8 April 1998 (1998-04-08) page 5, line 24 -page 7, line 18 figures 3-6 -----	1-19

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 01/11931

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6120549 A	19-09-2000	NONE	
EP 0834823 A	08-04-1998	US 5963454 A	05-10-1999
		EP 0834823 A1	08-04-1998
		JP 10198726 A	31-07-1998